

TMVA Fast Inference System (SOFIE)

ROOT Workshop 11.05.2022

*Sitong An, Lorenzo Moneta, Sanjiban Sengupta, Ahmat Hamdan, Federico Sossai,
Aaradhya Saxena*





Idea for Inference Code Generation

► An inference engine that...

- Input: trained ONNX model file
 - Common standard for ML models
 - Supported by PyTorch natively
 - Converters available for Tensorflow and Keras
- Output: Generated C++ code that hard-codes the inference function
 - Easily invokable directly from other C++ project (plug-and-use)
 - Minimal dependency (on BLAS only)
 - Can be compiled on the fly using Cling JIT



► **SOFIE : System for Optimised Fast Inference code Emit**



Code Generation

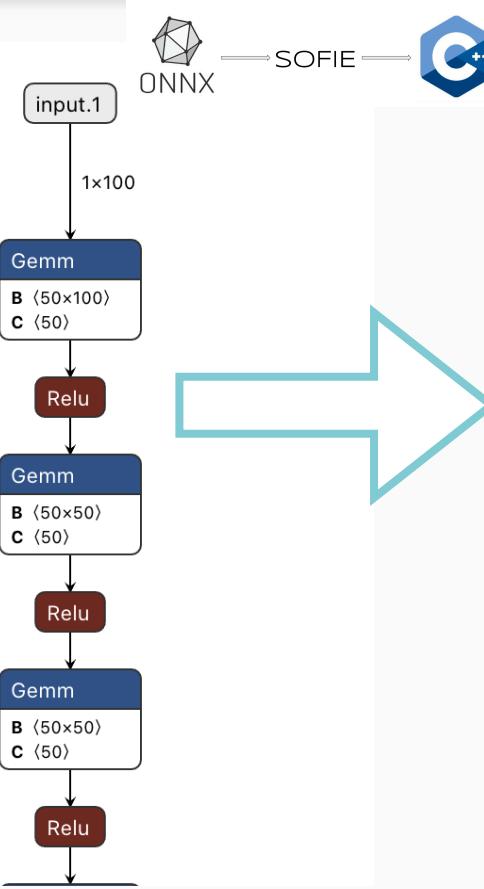
- ▶ Parser: from ONNX to an internal model representation:
TMVA::Experimental::SOFIE::RModel class

```
using namespace TMVA::Experimental::SOFIE;  
RModelParser_ONNX parser;  
RModel model = parser.Parse("model.onnx");
```

- ▶ Code Generation: from RModel to a C++ file (`model.hxx`)

```
// generate text code internally (with some options)  
model.Generate();  
// write output header file and data weight file  
model.OutputGenerated();
```

Code Generation



```
namespace TMVA_SOFIE_Linear_event{  
    struct Session {  
        Session(std::string filename = "Linear_event.dat") {  
            // read weight data file  
            std::ifstream f;  
            f.open(filename);  
            .....  
        }  
        std::vector<float> infer(float* tensor_input){  
            .....  
            //---- Gemm  
            BLAS::sgemm_(.....); // from tensor_input -> tensor_21  
            .....  
            //----- RELU  
            for (int i = 0; i < 50 ; i++)  
                tensor_22[i] = ((tensor_21[i] > 0 )? tensor_21[i] : 0);  
            .....  
            BLAS::sgemm_(.....);  
            .....  
            // return output tensor  
            std::vector<float> ret (tensor_39, tensor_39 + 10);  
            return ret;  
        };  
};
```

See tutorial [TMVA_SOFIE_ONNX.C](#)



SOFIE Libraries

- ▶ Separation between parser and code generation (RModel class)
 - ▶ ONNX Parser is in a separate library
`libROOTTMVASofieParser.so`
 - ▶ Dependency on Google Protocol Buffers (a.k.a. Protobuf) for parsing the ONNX file
 - ▶ RModel class and code generation is in another library:
`libROOTTMVASofie.so`
 - ▶ Minimal dependency on ROOT (for I/O)
 - ▶ Model can be serialised and stored in a ROOT file !
- ▶ Emitted C++ code requires only a linear algebra library (BLAS)

Minimal dependencies !



ONNX Supported Operators

Gemm	Implemented and integrated (ROOT 6.26)
Activations: Relu, Seul, Sigmoid,	Implemented and integrated
Convolution (1D, 2D and 3D)	Implemented and integrated
RNN, GRU, LSTM	Implemented and integrated
BatchNorm	Implemented and integrated
Pooling: MaxPool, AveragePool, GlobalAverage	Implemented and integrated
Layer operations: Add, Reshape, Flatten, Transpose, Squeeze, Unsqueeze, Slice	Implemented and integrated
Concat, Softmax, Leaky RELU, Identity, InstanceNorm, MatMul	Implemented but to be integrated (PR #8885, #9311, #8885, #10315)
Deconvolution	Planned for next release
???	Depending on user needs



Other SOFIE Parsers

- ▶ Parser exists also for :
 - ▶ Native PyTorch files (*model.pt* files)

```
SOFIE::RModel model = SOFIE::PyTorch::Parse("PyTorchModel.pt");
```
 - ▶ Native Keras files (*model.h5* files)

```
SOFIE::RModel model = SOFIE::PyKeras::Parse("KerasModel.h5");
```
- ▶ Based on the PyMVA interface (in `libPyMVA.so`)
 - ▶ Limited operator support:
only dense layer and convolutional layers
- ▶ See TMVA tutorials [TMVA_SOFIE_PyTorch.C](#) and [TMVA_SOFIE_Keras.C](#)



RDF Integration

- ▶ SOFIE Inference code provides a Session class with this signature:

```
vector<float> ModelName::Session::infer(float* input);
```

- ▶ RDF Interface requires a functor with this signature:

```
FunctorObj::operator()(T x1, Tx2, Tx3, ...);
```

- ▶ We have developed a generic functor adapting SOFIE signature to the RDF one
 - ▶ Support for multi-thread evaluation, using RDF slots

```
auto h1 = df.DefineSlot("DNN_Value",
SofieFunctor<7,TMVA_SOFIE_higgs_model_dense::Session>(nslots),
{"m_jj", "m_jjj", "m_lv", "m_jlv", "m_bb", "m_wbb", "m_wwbb"}).
Histo1D("DNN_Value");
```



SOFIE Functor for RDF

```
template <std::size_t... N, typename S, typename T>
struct SofieFunctorHelper<std::index_sequence<N...>, S, T> {
    // use index_sequence to define an operator () with N fixed parameter arguments
    ...
    // Constructor: create vector of Sessions
    SofieFunctorHelper(int nslots, const std::string & filename = "") {
        ...
        for (unsigned int i = 0; i < nslots; i++)
            fSessions.emplace_back(filename);
    }

    double operator()(unsigned slot, AlwaysT<N>... args) {
        fInput[slot] = {args...};
        auto y = fSessions[slot]->infer(fInput[slot].data());
        return y[0];
    }
};

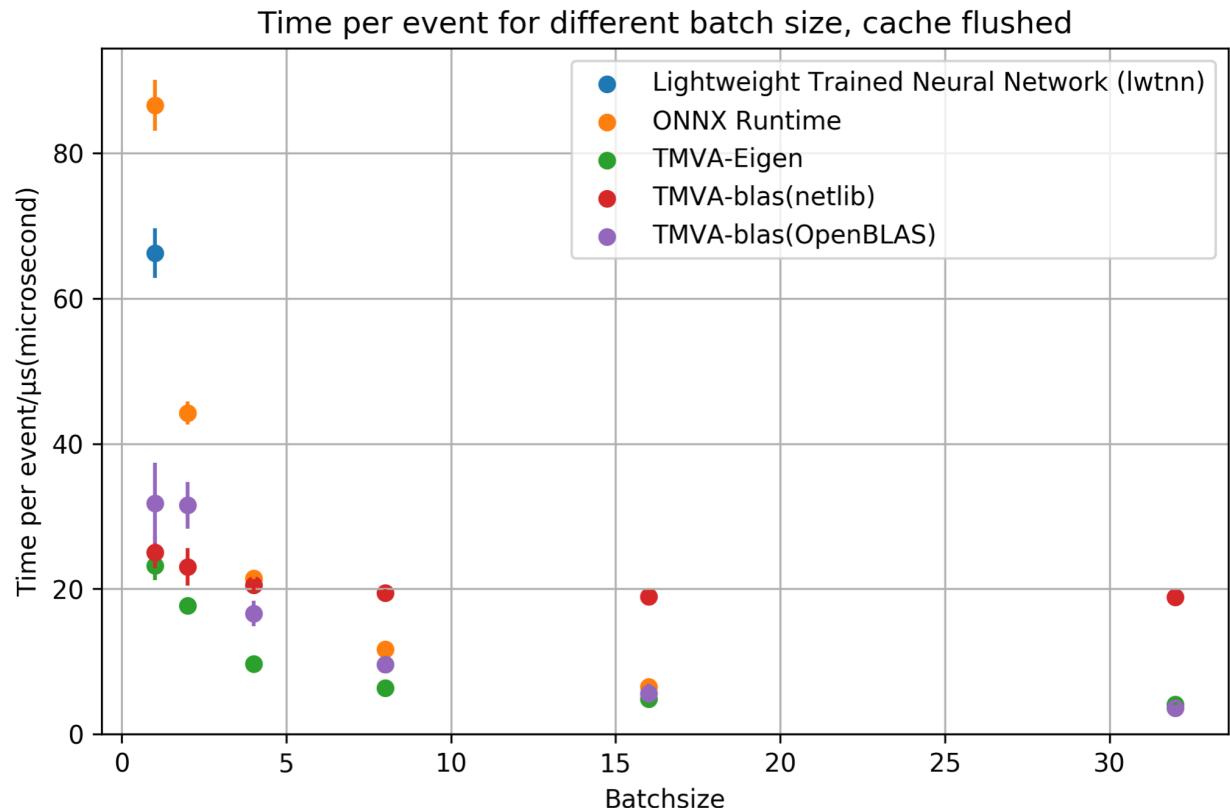
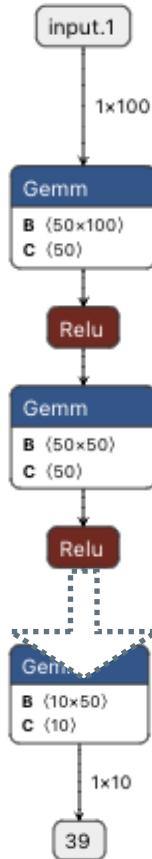
template <std::size_t N, typename F>
auto SofieFunctor(int nslot) -> SofieFunctorHelper<std::make_index_sequence<N>, F, float>
{
    return SofieFunctorHelper<std::make_index_sequence<N>, F, float>(nslot);
}
```

Full code available [here](#).



Benchmark: Dense Model

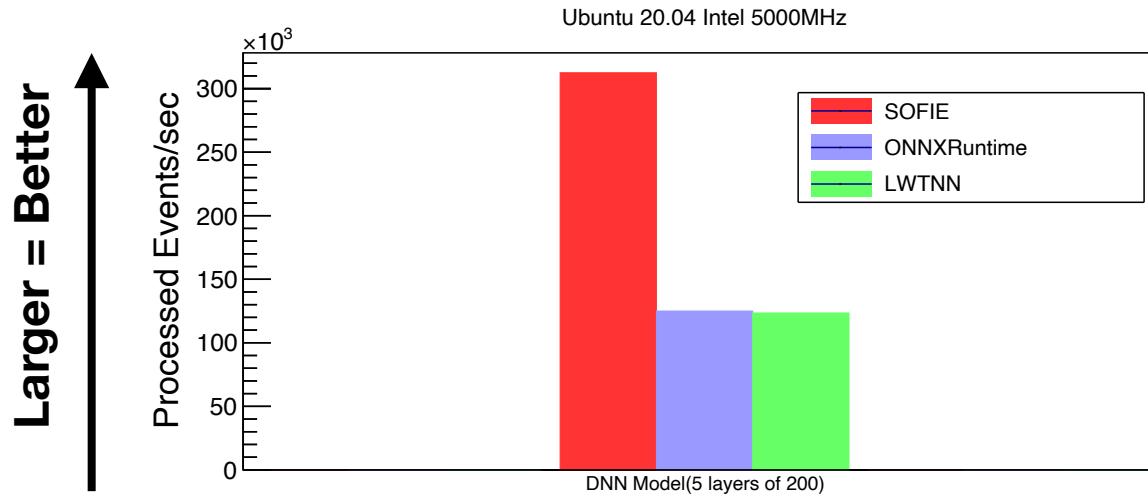
10 Dense layers





Benchmark with RDF

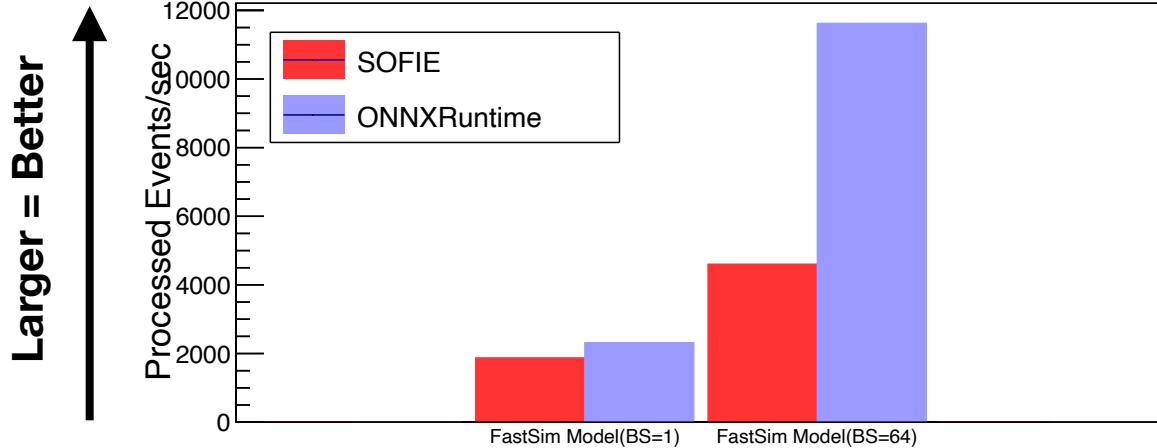
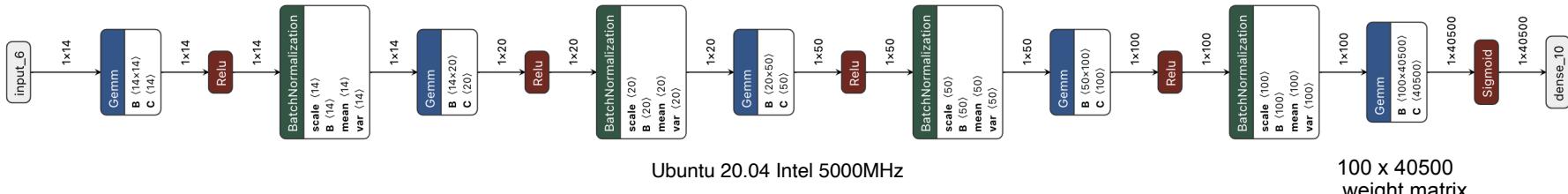
- ▶ Test on a Deep NN (from [TMVA_Higgs_Classification.C](#) tutorial, 5 FC layers of 200 units)
- ▶ Run on dataset of (5M events)
 - ▶ Single Thread, but can run Multi-Threads





Benchmark using a FastSim Model

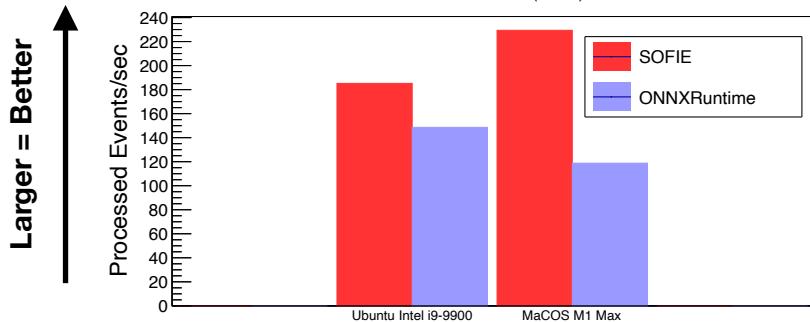
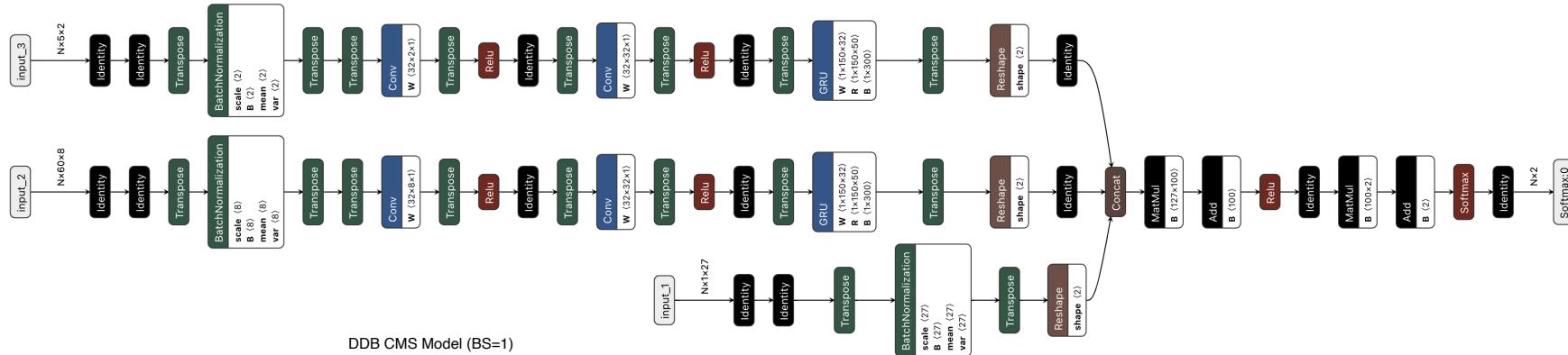
- ▶ Using ONNX model from a G4 example ([Par04](#))
 - ▶ dense layer with Relu and Batch normalization layers





Benchmark using a CMS Model

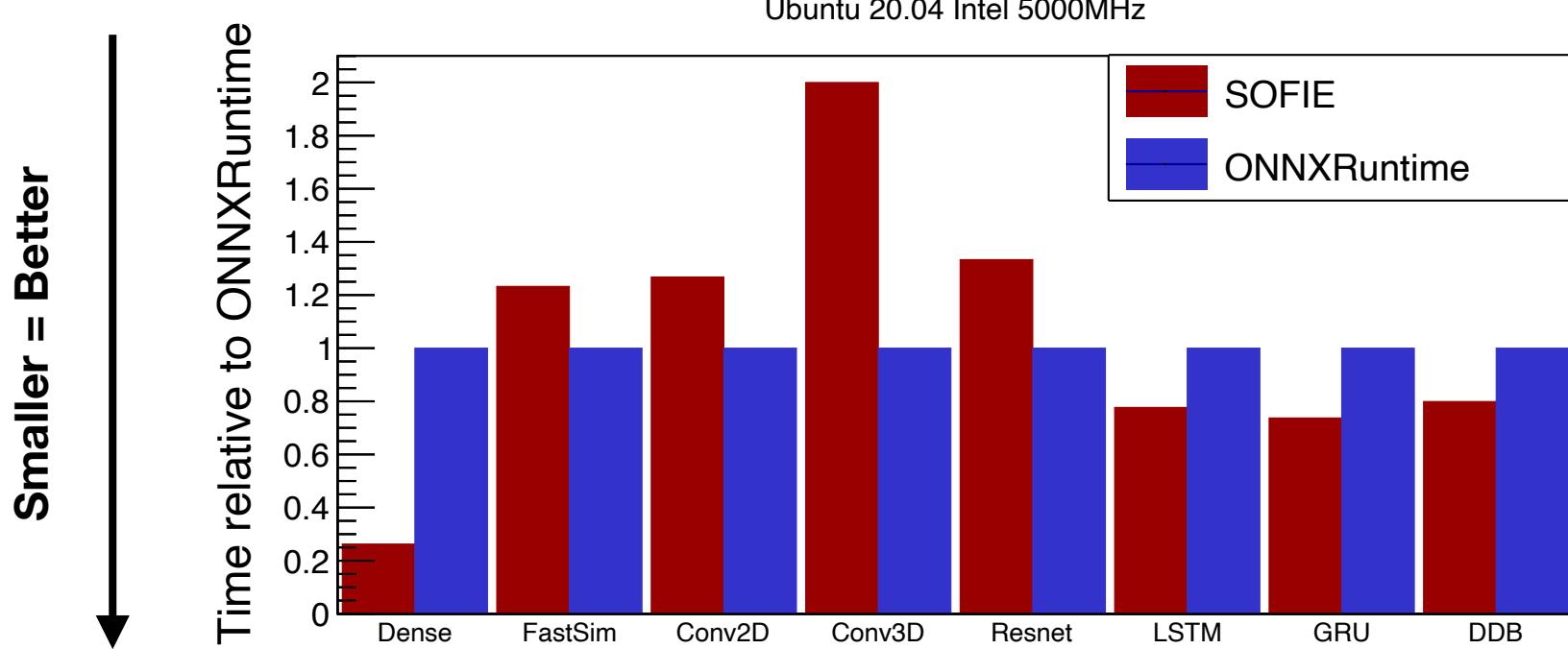
- ▶ Preliminary results using CMS Deep Double model (DDB.onnx)
 - ▶ 3 inputs with 1d Convolutions (32x32x1) + GRU (32x150 and 50x150)





Benchmark: All Models (on Linux PC)

- ▶ Test Event performance of SOFIE vs ONNXRuntime (BS=1)





Example Notebooks

- ▶ Some example notebooks on using SOFIE:
 - ▶ <https://github.com/lmoneta/tmva-tutorial/tree/master/sofie>
- ▶ Some tutorials are also available in the [tutorial/tmva](#) directory

- ▶ Implement some missing operators:
 - ▶ Deconvolution, etc..
 - ▶ more depending on user needs and feedback
- ▶ Improve parser for Keras models
- ▶ Integration with new RReader class for TMVA Inference
 - ▶ Have a user interface starting from an input model and performs internally JIT-ing of code
 - ▶ Example : **RBDT interface for fast BDT**

```
TMVA::RBDT bdt("myBDT", "model.root");
auto x = TMVA::RTensor<float>(data, shape);
auto y2 = bdt.Compute(x);
```



Other Possible Developments

- ▶ Implement further optimisations:
 - ▶ layer fusions, quantisations,....
- ▶ Generate code for different architectures (e.g GPU)
- ▶ Have SOFIE as an independent package
 - ▶ give possibility of frequent version updates
 - ▶ ROOT dependency for model serialisation could be optional
 - ▶ *if requested, it can be done*
- ▶ Store model weights in a binary file or other format
 - ▶ currently using a simple text file or including weights directly in header file declaration



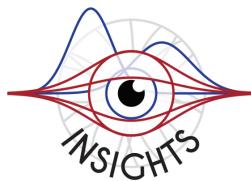
Summary

- ▶ First release of SOFIE, fast and easy to use inference engine for ML models, is available in ROOT 6.26
- ▶ Good performance compared to existing package (ONNXRuntime) and LWTNN
 - ▶ further optimisations are still possible
- ▶ Integrating with other ROOT tools (RDataFrame) and existing TMVA RReader class
- ▶ Future developments will be done according to user needs and the received feedback!



Conclusion

- ▶ [Link](#) to SOFIE in current ROOT master
- ▶ [Link](#) to TMVA/SOFIE tutorials
- ▶ [Link](#) to SOFIE notebooks
- ▶ [Link](#) to benchmark in rootbench (PR #239)
- ▶ [Link](#) to previous benchmark sample code



The presenter gratefully acknowledges the support of the Marie Skłodowska-Curie Innovative Training Network Fellowship of the European Commission Horizon 2020 Programme, under contract number 765710 INSIGHTS.